RSP — Technical Documentation

RSP — Technical Documentation for Developers

Overview

This document provides comprehensive technical documentation for developers who want to: 1. Understand the RSP protocol architecture 2. Integrate their project with RSP using the Revenue Connector SDK 3. Build on top of the RSP protocol

Prerequisites: Familiarity with Solidity, ERC-20 tokens, and Ethereum smart contract development.

Table of Contents

1. Architecture Overview
2. Smart Contract Reference
3. Integration Guide
4. SDK Reference
5. Deployment Guide
6. Security Considerations
7. API Reference (Contract ABIs)
8. FAQ

1. Architecture Overview

System Diagram

```
┌─────────────────────────────────────────────────────────────────┐
│                        RSP PROTOCOL                             │
│                                                                 │
│  RSP Token ──────── balanceOf() ────────→ RevenuePool           │
│     │                                        │                  │
│     ├── mint() ←── FairLaunch (max 30M)      ├── 1% fee → Treasury
│     ├── mint() ←── Emission (max 40M)        ├── 99% → Holders │
│     │                                        │                  │
│     ├── balanceOfAt() ←── Governance         ├── receiveRevenue()
│     │                        │               │       ↑          │
│     │                        ├── execute() ──→ ProjectRegistry  │
│     │                        │               │                  │
│     │                        ├── recordVoter() → Treasury       │
│     │                        └── recordVote() → Emission        │
│     │                                                           │
│     └── _beforeTokenTransfer() → RevenuePool._updateReward()   │
└─────────────────────────────────────────────────────────────────┘
                              ↑
                    IRevenueConnector
                              ↑
                    YOUR PROJECT HERE
```

Contract Addresses

(To be published after mainnet deployment)

Contract	Address	Etherscan
RSP Token	`TBD`	[Link]
FairLaunch	`TBD`	[Link]
RevenuePool	`TBD`	[Link]
Governance	`TBD`	[Link]
ProjectRegistry	`TBD`	[Link]
Treasury	`TBD`	[Link]
Emission	`TBD`	[Link]

---

2. Smart Contract Reference

2.1 RSP Token

Standard: ERC-20 + ERC20Votes (OpenZeppelin) **Supply:** 100,000,000 RSP (fixed, immutable) **Decimals:** 18

Key Functions

```
// Standard ERC-20
function transfer(address to, uint256 amount) external returns (bool);
function approve(address spender, uint256 amount) external returns (bool);
function transferFrom(address from, address to, uint256 amount) external returns (bool);
function balanceOf(address account) external view returns (uint256);
function allowance(address owner, address spender) external view returns (uint256);
function totalSupply() external view returns (uint256);

// ERC20Votes (for governance snapshot)
function delegate(address delegatee) external;
function getVotes(address account) external view returns (uint256);
function getPastVotes(address account, uint256 blockNumber) external view returns (uint256);

// Mint (restricted — only authorized minters)
function mint(address to, uint256 amount) external;
```

Events

```
event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);
event MinterAdded(address indexed minter, uint256 cap);
event MintingFinished();
event OwnershipRenounced(address indexed previousOwner);
```

Important Notes

- After `finishMinting()`, no new tokens can ever be created
- After `renounceOwnership()`, no admin functions can be called
- `_beforeTokenTransfer` hook calls `RevenuePool._updateReward()` on every transfer

2.2 FairLaunch

Purpose: Fixed-price RSP token sale

Key Functions

```
// Buy RSP with USDC
function buyWithUSDC(uint256 usdcAmount) external;

// Buy RSP with USDT
function buyWithUSDT(uint256 usdtAmount) external;

// View functions
function tokensRemaining() external view returns (uint256);
function tokensSold() external view returns (uint256);
function isActive() external view returns (bool);
```

Parameters

Parameter	Value
Price	0.10 USD per RSP
Total for Sale	30,000,000 RSP
Minimum Purchase	10 USD
Maximum Purchase	No limit
Accepted Tokens	USDC, USDT (native Ethereum)
Revenue Split	33% Treasury, 66% Liquidity Pool

Events

```
event TokensPurchased(address indexed buyer, address stablecoin, uint256 usdAmount, uint256
tokenAmount);
event SaleFinalized(uint256 totalTokensSold, uint256 totalUSDRaised);
```

2.3 RevenuePool

Purpose: Collects revenue from connected projects and distributes to token holders

Key Functions

```
// Called by connected projects to deposit revenue
function receiveRevenue(uint256 amount) external;

// Claim accumulated rewards
function claim() external;

// View functions
function earned(address user) external view returns (uint256);
function rewardPerToken() external view returns (uint256);
function getCirculatingSupply() external view returns (uint256);
function totalRevenueReceived() external view returns (uint256);
function totalDistributed() external view returns (uint256);
function totalFeeCollected() external view returns (uint256);
```

Revenue Flow

```
Project calls receiveRevenue(amount)
  → 1% fee transferred to Treasury
  → 99% added to reward pool
  → rewardPerTokenStored updated
  → Holders can claim proportional share
```

Circulating Supply Calculation

```
circulatingSupply = totalSupply
  - balanceOf(vestingContract)
  - balanceOf(uniswapLP)
  - balanceOf(fairLaunchContract)
  - balanceOf(emissionContract)
  - balanceOf(treasury)
  - balanceOf(revenuePool)
```

Events

```
event RevenueReceived(address indexed project, uint256 totalAmount, uint256 fee, uint256 distributed);
event RewardClaimed(address indexed user, uint256 amount);
```

2.4 Governance

Purpose: Decentralized decision-making for project approvals and treasury spending

Key Functions

```
// Submit a proposal (requires >= 1,000,000 RSP)
function propose(
    ProposalType proposalType,  // ProjectAdd or TreasurySpend
    bytes calldata callData,
    string calldata description
) external returns (uint256 proposalId);

// Cast a vote
function castVote(uint256 proposalId, VoteType voteType) external;

// Execute a passed proposal (after timelock)
function execute(uint256 proposalId) external;

// Cancel a proposal (only proposer, only while active)
function cancel(uint256 proposalId) external;

// View functions
function getProposal(uint256 proposalId) external view returns (Proposal memory);
function state(uint256 proposalId) external view returns (ProposalState);
function hasVoted(uint256 proposalId, address voter) external view returns (bool);
function quorumReached(uint256 proposalId) external view returns (bool);
function proposalCount() external view returns (uint256);
```

Immutable Parameters (Hardcoded)

```
uint256 public constant QUORUM_PERCENTAGE = 10;        // 10%
uint256 public constant VOTING_PERIOD = 20 days;        // 20 days
uint256 public constant TIMELOCK_DURATION = 72 hours;    // 72 hours
uint256 public constant PROPOSAL_THRESHOLD = 1_000_000e18; // 1M RSP
```

These values CANNOT be changed by any means.

Enums

```
enum ProposalType { ProjectAdd, TreasurySpend }
enum VoteType { For, Against, Abstain }
enum ProposalState { Active, Defeated, Succeeded, Timelock, Executable, Executed, Canceled }
```

Events

```
event ProposalCreated(uint256 indexed id, address proposer, ProposalType proposalType, string
description, uint256 snapshotBlock, uint256 startTime, uint256 endTime);
event VoteCast(uint256 indexed proposalId, address indexed voter, VoteType voteType, uint256
votingPower);
event ProposalExecuted(uint256 indexed id, address executor);
event ProposalCanceled(uint256 indexed id);
```

2.5 ProjectRegistry

Purpose: Registry of connected projects with automatic deactivation

Key Functions

```
// Register a project (only callable by Governance)
function registerProject(
    address contractAddress,
    address revenueConnector,
    string calldata name,
    string calldata description,
    uint256 revenueSharePercent,
    uint256 proposalId
) external;

// Record revenue (only callable by RevenuePool)
function recordRevenue(address projectAddress, uint256 amount) external;

// Check and deactivate if 90 days without revenue (callable by anyone)
function checkAndDeactivate(uint256 projectId) external;
function batchCheckAndDeactivate(uint256[] calldata projectIds) external;

// View functions
function getProject(uint256 projectId) external view returns (Project memory);
function isActive(address projectAddress) external view returns (bool);
function isRegistered(address projectAddress) external view returns (bool);
function getActiveProjects() external view returns (uint256[] memory);
function projectCount() external view returns (uint256);
function activeProjectCount() external view returns (uint256);
function daysSinceLastRevenue(uint256 projectId) external view returns (uint256);
```

Events

```
event ProjectRegistered(uint256 indexed id, address contractAddress, address revenueConnector, string
name, uint256 proposalId);
event ProjectDeactivated(uint256 indexed id, address contractAddress, uint256 daysSinceLastRevenue);
event ProjectReactivated(uint256 indexed id, address contractAddress);
event RevenueRecorded(uint256 indexed id, address contractAddress, uint256 amount, uint256
totalContributed);
```

2.6 Treasury

Purpose: Protocol treasury with multisig-to-DAO migration

Key Functions

```
// Transfer funds (multisig or governance depending on mode)
function transfer(address to, uint256 amount) external;
function transferETH(address to, uint256 amount) external;
function transferERC20(address token, address to, uint256 amount) external;

// Record voter for DAO migration tracking (only callable by Governance)
function recordVoter(address voter, uint256 votingPower) external;

// Check and execute migration if thresholds met (callable by anyone)
function checkMigration() external;

// View functions
function getBalance() external view returns (uint256);
function getETHBalance() external view returns (uint256);
function isDAOControlled() external view returns (bool);
function uniqueVoters() external view returns (uint256);
function totalVotingTokensUsed() external view returns (uint256);
function votersUntilMigration() external view returns (uint256);
function tokensUntilMigration() external view returns (uint256);
function migrationProgress() external view returns (uint256 voterPercent, uint256 tokenPercent);
```

DAO Migration Thresholds (Immutable)

```
uint256 public constant MIGRATION_VOTER_THRESHOLD = 1_000_000;      // 1M unique voters
uint256 public constant MIGRATION_TOKEN_THRESHOLD = 25_000_000e18;   // 25M RSP voting power
```

Both conditions must be met simultaneously. Migration is automatic, one-way, and irreversible.

Events

```
event TreasuryTransfer(address indexed to, uint256 amount, address token, string mode);
event NewVoterRecorded(address indexed voter, uint256 totalUniqueVoters);
event DAOMigrationExecuted(uint256 totalUniqueVoters, uint256 totalVotingTokensUsed, uint256
timestamp);
event FundsReceived(address indexed from, uint256 amount, address token);
```

2.7 Emission

Purpose: Quarterly distribution of RSP tokens to LP providers and governance voters

Key Functions

```
// Start next emission cycle (callable by anyone when previous cycle ended)
function startNextCycle() external;

// Distribute rewards for completed cycle (callable by anyone)
function distributeLPRewards() external;
function distributeVoterRewards() external;

// Claim rewards
function claimLPReward() external;
function claimVoterReward() external;
function claimAll() external;

// Record vote for voter rewards (only callable by Governance)
function recordVote(address voter, uint256 votingPower, uint256 cycleId) external;

// View functions
function currentCycle() external view returns (uint256);
function cycleTimeRemaining() external view returns (uint256);
function totalMinted() external view returns (uint256);
function totalRemaining() external view returns (uint256);
function earnedLP(address user) external view returns (uint256);
function earnedVoter(address user) external view returns (uint256);
function isFinished() external view returns (bool);
function cyclesRemaining() external view returns (uint256);
```

Parameters (Immutable)

```
uint256 public constant TOTAL_EMISSION = 40_000_000e18;    // 40M RSP
uint256 public constant CYCLE_LENGTH = 90 days;             // Quarterly
uint256 public constant TOTAL_CYCLES = 16;                  // 4 years
uint256 public constant TOKENS_PER_CYCLE = 2_500_000e18;    // 2.5M per quarter
uint256 public constant LP_SHARE = 60;                      // 60% to LP providers
uint256 public constant VOTER_SHARE = 40;                   // 40% to voters
```

Events

```
event CycleStarted(uint256 indexed cycleId, uint256 startTime, uint256 endTime);
event LPRewardsDistributed(uint256 indexed cycleId, uint256 amount, uint256 providerCount);
event VoterRewardsDistributed(uint256 indexed cycleId, uint256 amount, uint256 voterCount);
event LPRewardClaimed(address indexed user, uint256 amount);
event VoterRewardClaimed(address indexed user, uint256 amount);
event EmissionFinished(uint256 totalMinted, uint256 timestamp);
```

---

3. Integration Guide

Step 1: Install the SDK

```
npm install @rsp/sdk
```

Step 2: Choose a Template

```
// Available templates:
import { CasinoConnector } from '@rsp/sdk/templates';
import { GameConnector } from '@rsp/sdk/templates';
import { MarketplaceConnector } from '@rsp/sdk/templates';
import { PaymentConnector } from '@rsp/sdk/templates';
import { DeFiConnector } from '@rsp/sdk/templates';
import { SubscriptionConnector } from '@rsp/sdk/templates';
import { GenericConnector } from '@rsp/sdk/templates';
```

Step 3: Implement the Revenue Connector

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@rsp/sdk/contracts/base/RevenueConnectorBase.sol";

contract MyProjectConnector is RevenueConnectorBase {

    constructor(
        address _revenuePool,
        address _rspToken,
        address _uniswapRouter
    ) RevenueConnectorBase(_revenuePool, _rspToken, _uniswapRouter) {}

    // Implement: how does your project collect revenue?
    function _collectRevenue() internal override returns (uint256) {
        // Your logic here
        // Return the amount of revenue collected (in your native token)
        uint256 revenue = myProject.withdrawRevenue();
        return revenue;
    }
}
```

Step 4: Choose a Distribution Strategy

```
// Option A: On every transaction
modifier distributeOnTransaction() {
    _;
    if (pendingRevenue() > 0) {
        distributeRevenue();
    }
}

// Option B: When threshold is reached
modifier distributeIfThreshold(uint256 threshold) {
    _;
    if (pendingRevenue() >= threshold) {
        distributeRevenue();
    }
}

// Option C: Periodic (called externally, e.g., by a keeper)
function triggerDistribution() external {
    require(pendingRevenue() > 0, "No revenue");
    distributeRevenue();
}
```

Step 5: Test

```
npx hardhat test
npx rsp-sdk verify-connector --address <your-connector-address> --network sepolia
```

Step 6: Deploy (Immutable!)

```
npx hardhat deploy --network mainnet
# WARNING: Contract must be immutable. No proxy patterns allowed.
```

Step 7: Submit Governance Proposal

Either hold 1,000,000 RSP yourself, or find a holder willing to submit the proposal on your behalf.

The proposal must include: - Project smart contract address - Revenue Connector address - Project name and description - Revenue share percentage

---

4. SDK Reference

RevenueConnectorBase

```
abstract contract RevenueConnectorBase is IRevenueConnector {

    address public immutable revenuePool;
    address public immutable rspToken;
    address public immutable uniswapRouter;

    // Override this to define how revenue is collected
    function _collectRevenue() internal virtual returns (uint256);

    // Collects revenue, swaps to RSP, sends to RevenuePool
    function distributeRevenue() external;

    // Returns pending (undistributed) revenue
    function pendingRevenue() external view returns (uint256);

    // Internal: swaps collected tokens to RSP via Uniswap
    function _swapToRSP(uint256 amount) internal;
}
```

SwapModule

```
library SwapModule {
    // Swap ETH to RSP
    function swapETHToRSP(address router, uint256 ethAmount, uint256 minOut) internal returns
(uint256);

    // Swap ERC-20 to RSP
    function swapTokenToRSP(address router, address token, uint256 amount, uint256 minOut) internal
returns (uint256);

    // Get TWAP price for slippage calculation
    function getTWAPPrice(address pool, uint32 period) internal view returns (uint256);
}
```

Verification Script

```
# Verify your connector meets all requirements
npx rsp-sdk verify-connector --address <address> --network <network>

# Output:
#  IRevenueConnector interface implemented
#  RevenuePool address correct
#  RSP token address correct
#  Contract is not a proxy (immutable)
#  distributeRevenue() callable
#  pendingRevenue() returns value
#  Swap route functional
#  No admin functions that modify revenue flow
#
# Result: PASS  — Ready for governance proposal
```

---

5. Deployment Guide

For RSP Core Team

See `specs/08-architecture.md` for full deployment sequence.

For Project Developers

1. Deploy your Revenue Connector (immutable, no proxy)
2. Verify on Etherscan
3. Run `verify-connector` script
4. Submit governance proposal (or find a proposer)
5. Wait for voting (20 days) + timelock (72 hours)
6. Once executed, your project is live on RSP

---

6. Security Considerations

For Project Developers

- **Immutability Required:** Your Revenue Connector contract MUST be immutable. Upgradeable proxies will not be approved by governance.
- **Use SafeERC20:** Always use OpenZeppelin’s SafeERC20 for token transfers.
- **Slippage Protection:** Set appropriate slippage limits in swap functions.
- **TWAP Oracle:** Use TWAP (not spot price) for swap calculations.
- **Reentrancy Guard:** Protect all external-calling functions.
- **Audit:** We strongly recommend auditing your connector before proposing.
- **Test on Sepolia:** Full test on testnet before mainnet deployment.

Common Mistakes

1. Using upgradeable proxy → **Rejected by governance**
2. No slippage protection on swaps → **Sandwich attack vulnerability**
3. Hardcoding RSP token address wrong → **Revenue goes nowhere**
4. Not approving RSP token to RevenuePool → **receiveRevenue() reverts**
5. Unbounded loops in _collectRevenue() → **Out of gas**

---

7. API Reference (Contract ABIs)

Full ABI files will be published in the SDK package:

```
@rsp/sdk/
├── abis/
│   ├── RSPToken.json
│   ├── FairLaunch.json
│   ├── RevenuePool.json
│   ├── Governance.json
│   ├── ProjectRegistry.json
│   ├── Treasury.json
│   └── Emission.json
```

JavaScript Usage

```javascript
import { ethers } from 'ethers';
import RSPTokenABI from '@rsp/sdk/abis/RSPToken.json';
import RevenuePoolABI from '@rsp/sdk/abis/RevenuePool.json';

const provider = new ethers.BrowserProvider(window.ethereum);
const signer = await provider.getSigner();

// Read RSP balance
const rspToken = new ethers.Contract(RSP_TOKEN_ADDRESS, RSPTokenABI, provider);
const balance = await rspToken.balanceOf(userAddress);

// Claim revenue rewards
const revenuePool = new ethers.Contract(REVENUE_POOL_ADDRESS, RevenuePoolABI, signer);
const earned = await revenuePool.earned(userAddress);
const tx = await revenuePool.claim();
await tx.wait();
```

---

8. FAQ

General

Q: Can I connect any type of project? A: Yes — casino, game, marketplace, payment gateway, DeFi protocol, SaaS, or anything that generates revenue. The community decides through governance.

Q: What if my project doesn't generate revenue in RSP? A: The Revenue Connector SDK handles automatic swapping from your native revenue token (ETH, USDC, etc.) to RSP via Uniswap.

Q: What percentage of my revenue do I need to share? A: You decide. There is no minimum or maximum. The governance proposal includes your revenue share percentage, and the community votes on whether it’s acceptable.

Q: What happens if my project stops generating revenue? A: After 90 days without revenue, your project is automatically deactivated. If revenue resumes, it’s automatically reactivated. No penalty.

Technical

Q: Can I upgrade my contract after connecting? A: No. RSP requires immutable contracts. This is a security feature, not a limitation.

Q: What’s the gas cost of distributeRevenue()? A: Approximately 100,000-200,000 gas depending on the swap route. With threshold or periodic distribution strategies, this cost is amortized.

Q: Can I use a custom swap route? A: Yes. The SDK’s SwapModule supports custom routes. You can override `_swapToRSP()` in your connector.

Q: What Solidity version should I use? A: 0.8.20 or later. Required for built-in overflow protection.

Governance

Q: I don’t have 1,000,000 RSP. How do I submit a proposal? A: Find a holder with 1M+ RSP willing to submit on your behalf. The RSP community Discord is a good place to start.

Q: What if my proposal is rejected? A: You can resubmit after addressing the community’s concerns. There is no penalty for rejected proposals.

Q: Can my project be removed by governance vote? A: No. There is no “vote to remove” mechanism. Projects can only be deactivated automatically by the 90-day inactivity rule.

RSP — Revenue Sharing Protocol Technical Documentation v1.0 — March 2026